Docket No. 42P18828
Express Mail No. EV339914651US

UNITED STATES PATENT APPLICATION

FOR

SINGLE INSTRUCTION TYPE BASED HARDWARE PATCH CONTROLLER

Inventors:

Chee Siong Lee
Vui Yong Liew
Mikal C. Hunsaker
Michael N. Derr

Prepared by:

BLAKELY, SOKOLOFF, TAYLOR & ZAFMAN, LLP
12400 Wilshire Boulevard, Seventh Floor
Los Angeles, California 90025-1026
(310) 207-3800

SINGLE INSTRUCTION TYPE BASED HARDWARE PATCH CONTROLLER

BACKGROUND

Field

[0001] Embodiments of the invention relate to the field of a patch mechanism, and more specifically, to a system and method that can be used to detect and workaround defects and conditions existing in an integrated circuit chip.

Background

[0002] Typically, host processor communicates with various end-devices, such as hard drives, USB (Universal Serial Bus) devices and PCI (Peripheral Component Interconnect) add-on cards, via a chipset. The chipset may include one or more integrated circuit chips, such as a memory controller and an I/O (input/output) controller. As an integrated circuit chip, such as the I/O controller, is tested, errors or defects may be discovered in the chip. Conventionally, when an error or defect is detected in the integrated circuit, one of a number of approaches may be taken. One approach is to de-feature the functionality if possible. This approach reduces the value of the product and may often be unacceptable. Another approach is to workaround the problem through software settings if possible. In this approach, the BIOS modifies the hardware behavior as the system boots to avoid the problem in the future. However, the software settings that are currently available tend to be very specific and, therefore, typically not useful for complex defects in highly integrated silicon chip. A further approach is to workaround the problem through modified software algorithms if possible. This approach can impact performance greatly and is not available in many situations.

[0003] In computer systems, host processor generates request cycles, which are directed towards the end-devices. Generally, there are two categories of cycles, i.e. a write request cycle that is used to transport data from the host processor to the end-user device or a read request cycle that is used to read data from the end-user device. When a read request cycle is targeting a specific device, a successful read request would result in read data being returned as a completion packet to the requesting device, such as the processor. However, if there is a design error in the decode logic

of the end-device, this may cause the read request targeting it to be ignored. In such situations, a completion packet will not be returned to the processor that issued the read request and this may cause the processor to hang or freeze. Such problem may arise in situations, for example, where the cycle type accepted by a decode logic of the end-device is incorrect with respect to the specification. Conventionally, a hardware bug such as this may be fixed through changes made to the silicon design once the bug has been localized in the design. This approach, however, tends to be very expensive and requires long throughput to make the fix available.

BRIEF DESCRIPTION OF THE DRAWINGS

[0004] The invention is illustrated by way of example and not by way of limitation in the figures of the accompanying drawings in which like references indicate similar elements. It should be noted that the references to "an" or "one" embodiment of this disclosure are not necessarily to the same embodiment, and such references mean at least one.

[0005] Figure 1A shows a block diagram of one embodiment of a computer system in which a patch module may be implemented.

[0006] Figure 1B shows a block diagram of another embodiment of a computer system in which a patch module may be implemented.

[0007] Figure 2 shows a block diagram of a patch module according to one embodiment.

[0008] Figure 3 shows a state diagram of a state machine incorporated within a master control unit according to one embodiment.

[0009] Figure 4 shows a state diagram of a state machine incorporated within a sequencer according to one embodiment.

[00010] Figure 5 shows a block diagram of a cycle generator according to one embodiment.

[00011] Figure 6 shows a flowchart diagram of operations involved in programming the patch module according to one embodiment.

[00012] Figure 7 shows a flowchart diagram of operations performed by the patch module according to one embodiment.

DETAILED DESCRIPTION

[00013] In the following description, specific details are set forth. However, it is understood that embodiments of the invention may be practiced without these specific details. In other instances, well-known circuits, structures, techniques and requests have not been shown or described in detail to avoid obscuring the understanding of this description.

[00014] Figure 1A shows one embodiment of computer system 120 in which patch module 100 may be implemented. System 120 includes processor 102, memory108, chipset 104 and a number of devices 110-1 through 110-N. The devices 110 coupled to the chipset may include I/O devices, and other end-devices, such as storage devices, USB devices and PCI devices. Chipset 104 may include one or more integrated circuit chips, such as memory controller and I/O controller. In one embodiment, programmable patch module 100 is integrated into chipset 104 to enable a user to detect, repair and/or workaround a wide range of defects existing in the chipset. Patch module 100 may be used to workaround defects that are found after the production stepping has taped out. These workarounds could be distributed after going to production and platforms have been shipped out to the field, potentially avoiding a costly recall. A user may workaround a defect by using the patch module to ensure that the conditions that stimulate the defect can never exist.

[00015] Figure 1B shows another embodiment of computer system 150 in which patch module 100 is incorporated within I/O controller chip 158. System 150 includes processor 152, memory controller 154 and I/O controller 158. System 150 also includes memory 156 that is coupled to memory controller 154 and a number of devices 164-1 through 164-N coupled to I/O controller 158. In one context, I/O controller 158 may be any device or integrated circuit that connects devices 164 to an internal computer system bus, such as, for example, a PCI bus. I/O controller 158 includes host interface unit 160 that facilitates communication with processor 152. Host interface unit 160 provides incoming request cycles (posted request cycles and non-posted request cycles) to patch module 100. The term "non-posted request cycles" and "non-posted cycle" are used to refer to any request cycles that require completions, including, but not limited to, memory read requests, configuration read requests and configuration write requests. The term "posted request cycle" or "posted cycle" are used to refer to any request cycles that completes at the source before it

completes at the destination, such as, for example, memory write requests. In one embodiment, the incoming request cycles are received from memory controller 154 via a PCI Express interconnect coupled between I/O controller 158 and memory controller 154. I/O controller 158 also includes completion queue 162, which is coupled to patch module 100.

[00016] In operation, patch module 100 will sample incoming cycles and compare the cycle attributes with preprogrammed cycle attributes. Patch module 100 is capable of being programmed to perform various operations to enable a user to detect, repair and/or workaround a wide range of defects existing in I/O controller 158. In one embodiment, if a match is found, patch module 100 is capable of performing one or more of the following operations (1) insert timed delay, (2) insert conditional delay, (3) modify the request cycle that caused the patch trigger, (4) generate and send a new request cycle to an end-device, and (3) generate and send a completion cycle for a captured non-posted request cycle.

[00017] Embodiments of the present invention may be implemented in hardware or software, or a combination of both. However, embodiments of the invention may be implemented as computer programs executing on programmable systems comprising at least one processor, a data storage system (including volatile and non-volatile memory and/or storage elements), at least one input device, and at least one output device. Program code may be applied to input data to perform the functions described herein and generate output information. The output information may be applied to one or more output devices, in known fashion. For purposes of this application, a processing system includes any system that has a processor, such as, for example, a digital signal processor (DSP), a micro-controller, an application specific integrated circuit (ASIC), or a microprocessor.

[00018] The programs may be implemented in a high level procedural or object oriented programming language to communicate with a processing system. The programs may also be implemented in assembly or machine language, if desired. In fact, the invention is not limited in scope to any particular programming language. In any case, the language may be a compiled or interpreted language. The programs may be stored on a storage media or device (e.g., hard disk drive, floppy disk drive, read only memory (ROM), CD-ROM device, flash memory device, digital versatile disk (DVD), or other storage device) readable by a general or special purpose

programmable processing system, for configuring and operating the processing system when the storage media or device is read by the processing system to perform the procedures described herein. Embodiments of the invention may also be considered to be implemented as a machine-readable storage medium, configured for use with a processing system, where the storage medium so configured causes the processing system to operate in a specific and predefined manner to perform the functions described herein.

[00019] Figure 2 shows patch module 100 according to one embodiment. In one embodiment path module 100 is incorporated within I/O controller 158, as shown in Figure 1B. However, it should be noted that embodiments of the patch module, described herein, may be implemented in other types of integrated circuits. Patch module 100 generally includes trigger-matching logic 204 and control logic 212. In one embodiment, trigger-matching logic 204 includes cycle capture unit 206, trigger comparator 208, and trigger registers 210. Cycle capture unit 206 is coupled to receive incoming cycles from a requesting device, such as a processor, via an interface (e.g., host interface unit 160). Cycle capture unit 206 captures the incoming cycles and supplies information contained in the captured cycle to trigger comparator 208. Trigger comparator 208 compares the information supplied by cycle capture unit 206 with trigger conditions (e.g., preprogrammed cycle attributes) stored in trigger registers 210. In one embodiment, cycle capture unit is configured to generate handshake signals for capturing cycles from a bus (e.g., backbone bus) or host interface 160 included in I/O controller 158.

[00020] Trigger registers 210 are used to store a wide variety of trigger conditions or trigger events. The terms "trigger condition" and "trigger event" are used herein to describe any condition or event that computer system designers may wish to implement as conditions or activities that trigger the execution of the patch module. In one implementation, if it is desired to trigger on the presence of a non-posted cycle, trigger-matching logic 210 may be programmed, for example, by programming one or more of the trigger registers, to distinguish between non-posted cycles and posted cycles and provide a trigger signal to execute a set of instructions upon detection of a non-posted cycle. Other implementations are possible and contemplated. For example, trigger-matching logic may be programmed to trigger on

certain address ranges, specific byte addresses, I/O space information, configuration space information and/or memory space information.

[00021] Once trigger-matching logic 204 detects that the incoming cycle matches one or more of the conditions or events stored in trigger registers 210, trigger matching logic 204 will send a trigger signal 202 to control logic 212 to indicate which of the trigger condition(s) matched with the currently captured cycle. Based on this information, control logic is configured to select a set of patch operations to be performed by patch module 100. If however, the information contained in the incoming cycle does not match the trigger conditions or events specified in trigger registers 210, the incoming cycle will be forwarded to the destination device via a downstream bus.

[00022] In the illustrated embodiment, control logic 212 generally includes master control unit 214, sequencer 216, cycle generator 500, timed delay unit 218 and conditional delay unit 220. Master control unit includes state machine 300, instruction storage 228 and instruction select unit 230. Instruction storage 228 is used to store instructions that can be performed by the patch module. The instructions may be programmed into the instruction storage using any suitable programming technique. Instruction select unit 230 selects instructions from instruction storage 228 based on the trigger condition(s) that matched with the currently captured cycle as indicated by the trigger signal 202 transmitted by trigger-matching logic 204. Instruction select unit 230 sends the selected instructions to sequencer 216 so that it can executed the selected instructions. The operations of master control unit 214 will be described in more detail with reference to Figure 3. The operations of sequencer 216 will be described in more detail with reference to Figure 4. The operations of cycle generator 222 will be described in more detail with reference to Figure 5.

[00023] Figure 3 depicts one embodiment of state machine 300 included in the master control unit. State machine 300 includes idle state 310, patch busy state 320 and forward cycle state 330. State machine 300 remains in idle state 310 until an incoming cycle is received by the patch module via an internal bus residing in an integrated circuit chip, such as an I/O controller chip. When an incoming cycle is received, the trigger-matching logic determines if the information contained within the incoming cycle matches a trigger condition. If the incoming cycle causes a patch trigger, state machine 300 will transition from idle state 310 to patch busy state 320 as

shown by arrow 315. In patch busy state 320, the master control unit will select a set of instructions that are to be performed by the sequencer based on the matched trigger condition(s). Additionally, the master control unit will activate the sequencer to iterate through each patch operation that is associated with the trigger. When the last patch operation has been executed by the patch sequencer, state machine 300 will transition from patch busy state 320 to idle state 310 as shown by arrow 325. If the incoming cycle does not cause a patch trigger, state machine 300 will transition from idle state 310 to forward cycle state 330 as shown by arrow 335. In forward cycle state 330, the master control unit will inform the downstream cycle transmit control unit (shown in Figure 2) to send the captured cycle to its intended destination device. State machine 300 will return to idle state 310 when the captured cycle has been forwarded to the downstream cycle transmit control unit as shown by arrow 345.

[00024] The sequencer is responsible for stepping through the instruction sequence selected by the master control unit. Figure 4 depicts one embodiment of state machine 400 included in the sequencer. State machine 400 includes opidle state 410, opdecode state 420, opexe state 430, waitsmiac state 450 and opdone state 440. State machine 400 starts in opidle state 410. The sequencer is responsible for iterating through each patch operation that is associated to a trigger. In one embodiment, the sequencer is configured to enable a set of patch operations to be sequenced back-to-back per trigger. As discussed above, when master control state machine 300 is in patch busy state 320, master control unit will activate sequencer state machine 400 by transitioning from opidle state 410 to opdecode state 420 as shown by arrow 415. During opdecode state 420, the first instruction is loaded and decoded by the sequencer to determine the type of patch operation to be performed. If there is no synchronous SMI (system management interrupt) associated with the instruction, the sequencer will transition from opdecode state 420 to opexe state 430 as shown by arrow 425.

[00025] In opexe state 430, the instruction loaded and decoded by the sequencer is executed, to perform the operation indicated by the instruction. If the currently loaded instruction is not the last instruction in the instruction sequence, the sequencer will transition back to opdecode state 420 as shown by arrow 435. In opdecode state 420, the next instruction will be loaded and decoded. This process of going back and forth between opdecode state 420 and opexe 430 repeats until the last operation has

been completed. Once the execution of the last operation in the instruction sequence has been completed, state machine 400 will transition to the opdone state 440 as shown by arrow 437 and automatically transition to opidle state 410 as shown by arrow 465.

[00026] In opdecode state 420, if state machine 400 determines that a synchronous SMI is associated with the currently loaded instruction, the sequencer will transition from opdecode state 420 to waitsmiac state 450, as shown by arrow 445, which causes the operation to be delayed until the corresponding synchronous SMI has been acknowledged to indicate that the currently loaded patch operation can make progress. Accordingly, when the synchronous SMI has been acknowledged, state machine 400 will transition from waitsmiac state 450 to opexe state 430 as shown by arrow 455. A SMI signal may be asserted to a processor to alert the processor that a SMI event has occurred. The SMI signal may be asserted for any of a large number of possible reasons. For example, the SMI signal may be asserted if a system resource seeks access to a certain range of memory or to a particular address.

[00027] In one embodiment, each patch operation is defined via a single 32-bit instruction. Table 1 describes the fields of 32-bit instruction according to one embodiment. Each 32-bit instruction includes a number of fields that contain information necessary for performing a patch operation as specified by the instruction.

TABLE 1

32-bit Instruction	
Field Name	Description
Opcode type	This field identifies the type of operation to be performed by the patch module. The types of operations identified by this field include, but not limited to, (1) timed delay operation, (2) conditional delay operation, (3) workspace register operation, (4) blocked register operation and (5) completion operation.
Forward flag	When this flag is set, the cycle generated by this instruction is forwarded to downstream bus.
Discard completion flag	When this flag is set, the completion queue is instructed to discard the completion associated with this request cycle so that the requestor (e.g., memory controller or host processor) does not receive the completion. This is needed for injected completion requested cycles (i.e., new cycles generated by the patch module).
Field select bits	This field indicates whether the instruction is operating on (1) cycle type, (2) address or (3) data.

32-bit Instruction

Field Name	Description
Operator type	This field determines how the operand will modify the field (cycle type, address or data) selected. The types of operators identified by this field include, but not limited to, (1) Use the operand entry unmodified, (2) AND the selected register/field with the operand entry, (3) OR the selected register/field with the operating entry, and (4) XOR the selected register/field with the operating entry.
Unmodified completion header	When this flag is set, the header section of the completion queue is loaded with header information from the original unmodified non-postable cycle as captured in the blocked cycle registers. When this flag is not set, the header section of the completion queue is loaded with modified header information associated with modified request that is generated in the workspace registers.
Operand index	This field selects the entry from the operand array to be used by the operator.
Delay field	This field specifies an amount of timed delay provided by the timed delay operation.
SMI flag	When this flag is set, an SMI signal is generated and sent from the patch module to a host processor.
Terminate flag	This field indicates whether or not this instruction is the last instruction in the patch instruction sequence.

[00028] Based on information contained in the 32-instruction, patch module 100 is capable of performing a wide range of patch operations to workaround various defects and conditions existing in I/O controller and/or other portions of computer system. In one embodiment, patch module 100, based on information contained in the 32-bit instruction, is capable of (1) inserting timed delay, (2) inserting conditional delay, (3) modifying the request cycle that caused the patch trigger, (4) generating and sending a new cycle via downstream bus, and (3) generating and sending a completion cycle for captured non-posted request cycle.

[00029] The type of operation to be performed by patch module is indicated in opcode type field, which may used to specify (1) timed delay operation, (2) conditional delay operation, (3) workspace register operation, (4) blocked register operation and (5) completion operation. Timed delay operation is used to provide a timed delay of a certain duration as specified by the delay field. Conditional delay operation is used to provide a delay until a particular internal state is reached, which may be indicated by an internal state signal. Workspace register operation is used to operate on one of workspace registers, which may be used as the starting cycle for modifying. Workspace register operation may be used for generating new injected

cycles (i.e., new cycles generated by the patch module) or used for modifying information stored in workspace register. Blocked register operation is used to operate on one of blocked registers, which may be used as the starting cycle for modifying. Completion operation is used to generate a completion packet for captured non-posted cycles and return the completion packet to the requesting device (e.g., host processor).

[00030] Figure 5 shows one embodiment of patch cycle generator 500 for executing a 32-bit instruction selected by the control logic of the patch module. The cycle generator is used to generate a modified cycle or a new cycle based on information contained in the currently loaded instruction. When an incoming cycle is received by the patch module, the information from the cycle that caused the patch module to trigger is stored in blocked cycle registers 504. In one embodiment, patch module employs three blocked cycle registers; namely cycle type register 504-1 is used to store cycle type related information associated with the captured cycle; address register 504-2 is used to store address information; and data register 504-3 is used to store data associated with the captured cycle. When the information contained in the blocked cycle registers is modified the resulting cycle is loaded into the workspace registers. There are also three workspace registers 502; namely cycle type register 502-1 is used to store cycle type information associated with the captured cycle; address register 502-2 is used to store address information; and data register 502-3 is used to store data associated with the captured cycle.

[00031] The workspace registers 502 are coupled to multiplexor 506, which selects one of the workspace registers (cycle type, address or data) to be sent to multiplexor 510 based on information supplied by the field select bits 528. The field select bits are contained within the 32-bit instruction and are used to indicate whether the instruction is operating on (1) cycle type register, (2) address register or (3) data register. Similarly, the blocked cycle registers 504 are coupled to multiplexor 508, which selects one of the blocked cycle registers (cycle type, address or data) to be sent to multiplexor 510 based on information supplied by the field select bits 528. Multiplexor 510 selects one of the workspace registers 502 and blocked cycle registers 504 to be sent to the rest of patch cycle generator 500 based on information supplied by opcode type field 530 of the instruction. Specifically, if the opcode type field 530 specifies that the workspace register operation is to be performed, one of the workspace registers 502 will be selected by multiplexor 510 to be sent to one of the

inputs of XOR gate 514, OR gate 516 and AND gate 518. On the other hand, if the opcode type field 530 specifies that the blocked register operation is to be performed, one of the blocked cycle registers 504 will be selected by multiplexor 510 to be sent to one of the inputs of XOR gate 514, OR gate 516 and AND gate 518.

[00032] The other input of the XOR gate 514, OR gate 516 and AND gate 518 is coupled to receive an operand entry from patch operand array 526. Patch operand array includes a number of operand entries. Patch cycle generator selects one of the operand entries to be used during the current operation based on information supplied by the operand index field of the instruction. The selected operand entry is used to modify the data contained in the register selected by multiplexor 510. In one embodiment, XOR gate, OR gate and AND gate are used to logically combined the selected register with the selected operand entry.

[00033] Patch cycle generator further includes multiplexor 520 to select one of the outputs of the XOR gate, AND gate and OR gate to be sent to multiplexor 524. One of the inputs of multiplexor 520 is coupled directly to patch operand array 526 to receive the selected entry. The output of multiplexor 520 is determined by operator type field 534 of the instruction. Thus, if one of XOR, OR or AND operation is indicated by the operator type field 534, then output from the appropriate gate is selected by multiplexor 520 to be sent the remainder of cycle generator 500. Otherwise, if the operator type field 534 indicates that the values in the selected operand entry is to be loaded unmodified, then input 540 (coupled directly to the patch operand array 526) is selected by multiplexor 520 to be sent to the remainder of cycle generator 500. The output of the multiplexor 520 is also coupled to the workspace registers 502 so that the resulting cycle can be loaded back into one of the workspace registers 502. Multiplexor 524 is used to determine whether or not the resulting cycle needs to be forwarded to downstream bus based on the forward flag of the instruction. Thus, multiplexor 524 will forward the resulting cycle to downstream bus if the forward flag is set.

[00034] Figure 6 shows a flowchart diagram of operations involved in programming the patch module according to one embodiment. In block 605, the buffer that holds the instruction sequence is programmed by software. In block 610, trigger registers residing within the trigger-matching logic are programmed to detect

trigger conditions. In block 615, the entries contained within the patch operand array are programmed by software.

[00035] Figure 7 shows a flowchart diagram of operations performed by the patch module according to one embodiment. In block 705, request cycles coming into an integrated circuit chip (i.e., I/O controller) are captured and relevant information from the header section the captured cycle is loaded into block cycle registers. In block 710, the information stored in the block cycle registers is compared with information stored in trigger registers. Patch module may be implemented with various trigger conditions, including, but not limited to, accesses to particular regions of memory space, configuration space or accesses to certain ports or devices. In block 715, the patch module determines whether the captured cycle matches one or more of the trigger conditions. In block 720, incoming cycles that do not cause a trigger are routed on to the rest of the chip (e.g., I/O controller). The patch module then returns to process the next incoming cycle. In block 725, if the incoming cycle matches a trigger condition, a sequence of instructions to be performed by the patch module is selected by the control logic based on the matched trigger condition.

[00036] Once the sequence of instructions has been selected, the patch module proceeds in a loop (blocks 730-760) to execute the sequence of instructions sequentially. In block 735, the execution of the current instruction may perform one of the following operations (1) generate and send a new cycle, (2) modify the request cycle that caused the patch trigger, (3) generate and send a completion cycle, and (4) insert delay. In block 740, the patch module is capable of generating and sending a new cycle. This may be useful in modifying and/or accessing the state of certain components or ports of the computer system. In block 745, the patch module is capable of modifying the incoming cycle that is captured. For example, if the trigger-matching logic detects an error condition, such as an incorrect address to a register, patch sequence could modify the address associated with the cycle and route it to a different location. In one embodiment, the patch module is capable of working around a failure that can be caused by a non-posted request cycle (e.g., read request) by capturing such non-posted request cycle and performing the necessary modification to the captured cycle before presenting it to the final destination. The read request will be modified properly such that it can be decoded correctly by the end-device and the

corresponding completion data can be returned to the requesting device (e.g., host processor).

[00037] In block 750, the patch module is capable of generating and returning completion to the requestor, which corresponds to the capture request cycle. This may be useful for avoiding failures caused by completion packet not being returned as a result of defect or condition existing in the system. In block 755, the patch module is capable of delaying the stream of incoming cycles. This may be useful for avoiding failures in which concurrent events are required. In block 760, once the patch device completes execution of each instruction in the patch sequence, it determines whether or not there are additional instructions in the sequence. If the sequence is not complete, then the next instruction is processed executed. If the sequence is complete, the patch device returns to process the next incoming cycle.

[00038] Embodiments of the patch module described herein may be used to workaround a wide range of defects and conditions. One example of a patch module usage is as follows. Suppose a defect exists in a computer system such that a read to a device (e.g., endpoint device) will return a wrong data unless a configuration setting is set or changed. In this case, the patch module can be programmed to trigger on that particular read. Once triggered, the patch module can be programmed to inject a configuration write (non-posted cycle) to temporarily change the device setting, then perform the original read with completion data returned back to the processor, followed by another configuration write to change the setting back to the original value. Another example of a patch module usage is as follows. Suppose a defect exists in a computer system such that a first read request to a particular device will not return a good data. In this case, the patch module can be programmed to trigger on the first read request to that particular device. Once triggered, the patch module can be programmed to inject the original read and discard the returned data. Then issue the same read request again and this time, return the good data back to the requesting device (e.g., processor). Although specific examples of how the patch module may be used are described above, it should be noted that embodiments of the patch module are not intended to be limited to these specific examples. Rather, the patch module is capable of being programmed in a wide variety of ways to workaround a wide range of defects and conditions existing within a computer system.

[00039] While several embodiments have been described, those skilled in the art will recognize that the invention is not limited to the embodiments described, but can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting.